ARCUS
Standard-Architektur
für Informationssysteme

Von Monolithen zu Komponenten

Wege zu objektorientierten Software-Architekturen

Zur Einreichung für die GI-Jahrestagung 1996

Dr. Peter Brössler,
Wolfgang Keller

München März 1996

sd&m
software design & management
GmbH & Co. KG
Thomas-Dehler-Straße 27
81737 München
Telefon (089) 6 38 12 - 210
Telefax (089) 6 38 12 - 490

Von Monolithen zu Komponenten
Wege zu objektorientierten Software-Architekturen

Peter Brössler, Wolfgang Keller
sd&m – software design & management GmbH & Co. KG
Thomas-Dehler-Str. 27 – 81737 München
Email: {Peter.Broessler, Wolfgang.Keller}@sdm.de

Abstract. *Verteilte objektorientierte Software-Architekturen und Wiederverwendung von Komponenten sind oft Ziele einer Einführungsstrategie für Objektorientierung bei Anwendern von Informationstechnologie. Die Ausgangsbasis sind jedoch, gerade bei Großanwendern, noch häufig monolithische, transaktionsbasierte Anwendungssysteme. Eine Stichtagsablösung ist riskant und auch meist nicht durchführbar. Der Artikel diskutiert daher einzelne Strategiemuster für den Weg von Monolithen zu objektorientierten Komponenten. Die einzelnen Muster können zu einem Ablösungsweg zusammengefügt werden, an dessen Ende eine durchgängig objektorientierte Software-Architektur stehen soll. Dabei wird davon ausgegangen, daß auf einem solchen Weg verschiedene Generationen von Softwaretechnologie noch lange koexistieren müssen.*

1 Einführung und Überblick

Einführungsstrategien für objektorientierte Technologien befassen sich meist mit der Einführung objektorientierter Modellierungs- und Programmierumgebungen [Gol+95, Gra95, Mey95] in einem Unternehmen. Am Ende des Prozesses soll eine objektorientierte Systemarchitektur und Entwicklungsumgebung stehen. Es wird jedoch meist nicht diskutiert, wie gerade große Unternehmungen ihre Investitionen in nicht-objektorientierte Software schrittweise in eine neue Welt verteilter Objekte und wiederverwendbarer Softwarekomponenten migrieren können. Am Anfang des Migrationsprozesses stehen meist monolithische, Host-basierte Transaktionssysteme unterschiedlicher Qualität. Mit Strategien für eine schrittweise Migration von Host-zentrierten Systemen zu verteilten, objektorientierten Software-Architekturen[1] befaßt sich dieser Artikel. Da große Organisationen betrachtet werden, sollen Strategien der Stichtagsablösung nicht diskutiert werden. Dies ist bei Großanwendern mit 200 und mehr Entwicklern selten realistisch [Ebe+94]. Dort ist eher damit zu rechnen, daß die Umstellung auf objektorientierte Arbeitsweisen 10 und mehr Jahre dauern kann. Während dieser Zeit müssen alte und neue Systeme kooperieren können.

Für eine schrittweise Migration kann dabei nicht nur eine mögliche Strategie gefunden werden. Vielmehr bieten sich, abhängig von den Gegebenheiten des Anwenders,

---

[1] Ob und wann dies überhaupt ein erstrebenswertes Ziel ist, soll hier nicht diskutiert werden. Dazu sei auf die einschlägige Literatur, zum Beispiel wieder [Gol+95, Gra95, Mey95], verwiesen.

verschiedene Ansatzpunkte für Verbesserungen, die sich zu einer Gesamtstrategie kombinieren lassen. Jede einzelne Strategie ist die Lösung für ein bestimmtes Migrationsproblem. Die Strategien werden hier deshalb in Anlehnung an Design-Patterns dargestellt [Cop+95, GOF95], da man so gezwungen wird, das Problem sowie die Vor- und Nachteile einer Lösung zu beschreiben[2].

Zunächst werden die verschiedenen Strategien kurz vorgestellt. Sie setzen an verschiedenen Stellen einer Anwendungs- [Den91] oder Client/Server-Architektur [Han95] an. Die Strategien werden dann mit ihren Vor- und Nachteilen einzeln diskutiert.

2 Migrationswege, Systemarchitektur und Bewertungskriterien

Eine Auswahl möglicher Migrationstrategien soll hier zunächst im Überblick anhand einer konventionellen Drei-Schichten-Architektur für betriebliche Informationssysteme [Den91, Bus+96] eingeführt werden.

2.1 Strategien

Offline-Systeme sind für uns parallel zu Altsystemen neu gebaute Systeme, die vollständig mit Objekttechnologie, meist auf einer Client-Plattform[3], realisiert werden, eine getrennte Datenhaltung besitzen und keine Online-Transaktionen im Zentralsystem vornehmen.

Unter der Strategie „***Alte Transaktionen mit neuem GUI***[4]" sollen Strategien verstanden werden, bei denen lediglich die Benutzungsoberfläche transaktionsorientierter Systeme ersetzt, nicht jedoch die Anwendungslogik ergänzt wird.

Die „***Migration mit CORBA***" bedeutet, daß der bestehende Anwendungskern von Systemen in Objekten gekapselt wird. Der Aufruf des Anwendungskerns erfolgt dann über eine Broker-Architektur [Bus+96].

Unter der Strategie „***objektorientierte Datenintegration***" wollen wir ein Vorgehen verstehen, bei dem alte und neue Anwendungen über gemeinsame Datenhaltung (meist hierarchische oder relationale Datenbestände) miteinander kommunizieren

---

2 Eine einzelne Beschreibung eines Patterns (zum Beispiel Broker bei [Bus+96]) kann den Umfang dieses Artikels sprengen. Wir haben uns daher bei der Form eher an Pattern Languages orientiert, wie sie in [Cop+95] beschrieben werden und die Zwischenüberschriften weggelassen.

3 Uns ist bewußt, daß in verteilten objektorientierten Systemen die Unterscheidung zwischen Clients und Servern nur noch eine logische und keine physische mehr ist. Trotzdem wollen wir der Einfachheit halber den Begriff Client für einen Arbeitsplatzrechner eines Endbenutzers und den Begriff Host oder Server für Host-basierte Unternehmensserver verwenden.

4 Graphical User Interface.

und parallel betrieben werden. Dabei wird für neue Softwarekomponenten eine objektorientierte Sicht auch auf alte Datenbestände geboten.

Es fällt auf, daß sich die Ansatzpunkte für die genannten Migrationsstrategien dort befinden, wo man auch Client/Server-Schnitte [Han95, Orf+94] ansetzen kann (Abbildung 1). Die Umsetzung der Strategien geht also meist mit der Installation einer Client/Server-Infrastruktur einher.

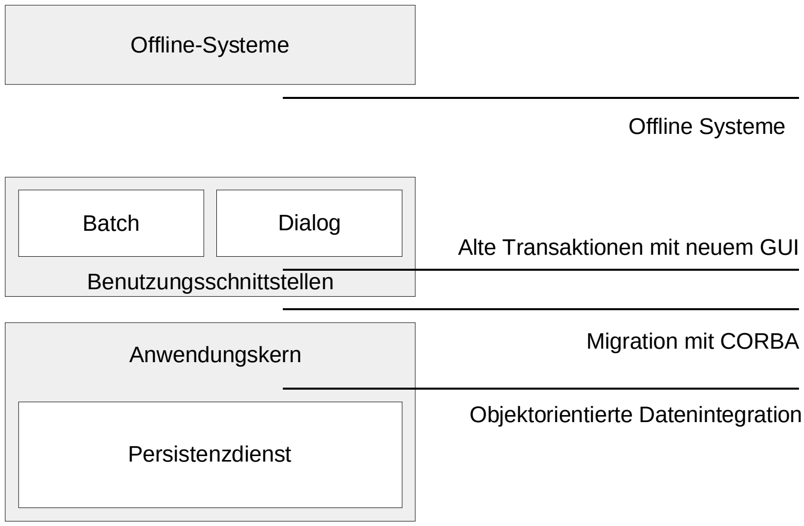

Abbildung 1: Migrationsstrategien und ihre Ansatzpunkte in einer Systemarchitektur

Eine Totalablösung von Altsystemen durch objektorientierte Systeme in einem einzigen Schritt wird hier nicht diskutiert werden. Bei den von uns betrachteten Großanwendern ist ein solches Vorgehen selten empfehlenswert [Ebe+94].

2.2 Bewertungskriterien

Will man verschiedene Lösungen für eine Problemfamilie beschreiben, so benötigt man eine Liste von Kriterien[5], mit denen man die Lösungen bewerten kann. Da die wesentlichen Kriterien für alle Migrationsstrategien gleich sind, wird die Liste hier zentral angegeben.

- ***Performance:*** Gerade bei Anwendern mit stark belasteten Transaktionssystemen darf die Performance neuer Anwendungen in der Wahrnehmung der Benutzer nicht wesentlich hinter der von existierenden Systemen zurückbleiben. Andernfalls wird die neue Technologie sofort diskreditiert.

---

5 In der Pattern-Gemeinde werden solche Kriterien auch *Forces* genannt. Die Bezeichnung geht auf die Arbeiten von Christopher Alexander zurück (siehe zum Beispiel [Ale79]), der das Finden einer Lösung als das Ausbalancieren von Kräften verstanden hat.

- ***Migrationskosten:*** Migrationsstrategien sollten so gewählt werden, daß die entstehenden Kosten minimiert werden oder zumindest durch entstehenden betriebswirtschaftlichen Nutzen aus verbesserter Systemfunktionalität gut zu rechtfertigen sind.
- ***Entkopplung:*** Wenn man davon ausgeht, daß objektorientierte und konventionelle Systementwicklung länger koexistieren müssen, muß darauf geachtet werden, daß sich beide nicht gegenseitig stören. Lösungen, bei denen eine ungestörte Parallelentwicklung stattfinden kann, sind daher solchen vorzuziehen, bei denen die Entwicklung objektorientierter Systeme die Weiterentwicklung konventioneller Systeme erschwert.
- ***Wartbarkeit und Redundanzfreiheit:*** Im Sinne besserer Wartbarkeit sind alle Lösungen negativ zu bewerten, bei denen Fakten redundant berücksichtigt werden müssen. Pathologisches Beispiel ist ein doppelter Anwendungskern, der im schlimmsten Fall auf Host und Client in unterschiedlichen Sprachen programmiert ist.
- ***Klarheit der Architektur:*** Es sind all jene Strategien positiv zu bewerten, mit denen früh eine klare, objektorientierte Architektur und Systemstrukturierung einziehen kann. Auf lange Frist negativ zu bewerten sind Korrekturen an der Benutzungsoberfläche, die jedoch schlechte Anwendungskernstrukturen unangetastet lassen.

Es gibt natürlich weitere Kriterien (Forces), die bei der Auswahl einer geeigneten Strategie zu beachten sind. Erkennbar ist schon hier, daß die durch die Kriterien gegebenen Ziele nicht konfliktfrei sind. Die in einem Unternehmen bevorzugte Lösung findet man durch Abwägen in Zielkonflikten und leider nur selten dadurch, daß man alle Ziele optimal erfüllen kann.

3 Offline-Systeme

Unter *Offline-Systemen* sollen hier solche Systeme verstanden werden, die auf einem replizierten Datenbestand arbeiten und parallel zu Altsystemen neu entwickelt werden. Sie können im Rahmen einer Migrationsstrategie meist vollständig mit Objekttechnologie realisiert werden. Typische Beispiele sind Außendienstsysteme [Har+94].

Offline-Systeme sind dann eine Wahl, wenn eine angestrebte Dezentralisierung von Unternehmensstrukturen mit einer Dezentralisierung der DV-Infrastruktur einhergehen soll. Dabei tritt zudem das Problem auf, daß vergleichsweise ungeschulte Außendienstmitarbeiter Systeme nur selten benutzen und daß Schulungskosten möglichst gering gehalten werden sollen. Die Anforderungen an eine Benutzeroberfläche sind also völlig andere als bei routinierten Sachbearbeitern.

Offline-Systeme bieten die Möglichkeit, mit Objekttechnologie graphische Benutzungsoberflächen zu realisieren. Dies geschieht auf einem eingeschränkten Replikat der zentralen Datenbasis. Man kann hier mit geringem Risiko zum Beispiel auch objektorientierte Datenhaltung [Cat94] in Pilotprojekten erproben und per Batch ver- und entsorgen (siehe Abbildung 2).

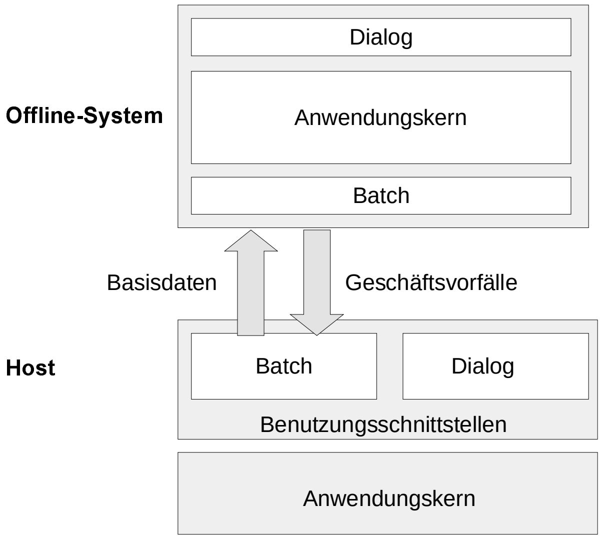

Abbildung 2: Offline-System mit Kopplung über Batch-Schnittstelle

Man findet bei Offline-Systemen noch Varianten. Je nach Anforderungen und Art der abzuwickelnden Geschäftsvorfälle hat man die Optionen, entweder mit einem Replikationsprodukt für Datenbanken oder mit einem Batch-Buchungssystem zu arbeiten, das offline erfaßte Geschäftsvorfälle als Stapel und mit demselben Anwendungskern wie das zentrale System verbucht. Dazu muß nicht immer auch ein eigener Batch auf dem Host erstellt werden. Das Client-System kann für Buchungen auch vorhandene Host-Transaktionen nutzen (Kombination mit Strategie *„Alte Transaktionen mit neuem GUI“*, siehe Abbildung 3).

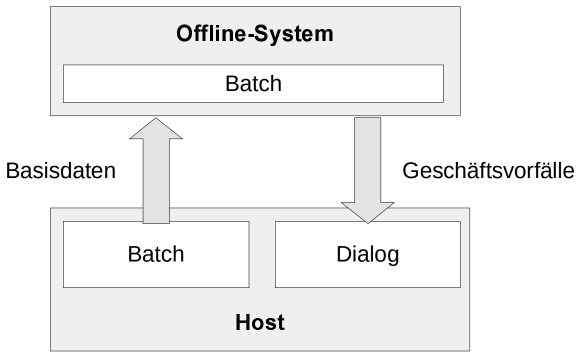

Abbildung 3: Offline-System benutzt Host-Dialoge zur Ausführung von Geschäftsvorfällen

Offline-Systeme bieten sich als OO-Pilotprojekte geradezu an. Sie sind von der herkömmlichen DV-Infrastruktur gut entkoppelt. Performance spielt bis auf den Datenabgleich eine unkritische Rolle, da es sich oft um Einbenutzersysteme handelt. Man kann durch die gute Entkopplung klare Architekturvorstellungen verfolgen. Eingriffe in ein Altsystem sind bis auf Batchläufe für den Datenabgleich selten erforderlich. Die Migrationskosten bewegen sich in einem Rahmen, der bei einer Dezentralisierung sowieso anfällt, da zeichenorientierte Oberflächen für „Gelegenheitsbenutzer" erhöhte Schulungs- und Benutzungskosten verursachen würden.

Wesentlicher Nachteil ist Redundanz. Dies bezieht sich nicht nur auf Daten, sondern vor allem auch auf redundante Anwendungskernfunktionalität, die parallel zum Altsystem entwickelt werden muß.

4 Alte Transaktionen mit neuem GUI

Unter der Strategie *„Alte Transaktionen mit neuem GUI"* werden Projekte verstanden, bei denen bestehende Host-Transaktionen (meist *Conversational Transactions* [Gra+93]) mit einer neuen, graphischen Oberfläche versehen werden. Die Präsentation erfolgt dabei auf dem Client.

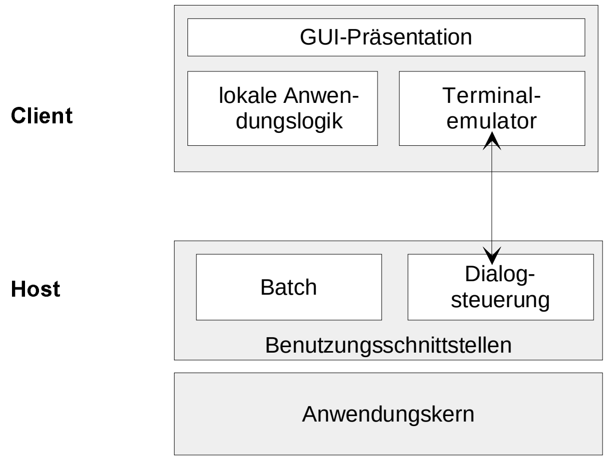

Abbildung 4: GUI-Client simuliert Terminal

Das zugrundeliegende Problem ist meist, daß bisher zentrale EDV-Funktionalität für Sachbearbeiter auch dezentral – also näher am Kunden – zur Verfügung gestellt werden soll. Das Ausgangsszenario ist dem für Offline-Systeme sehr ähnlich. Ein weiteres Anwendungsgebiet sind Buchungssysteme, die auch bei Kunden installiert werden sollen[6], statt ausschließlich zentral beim eigenen Unternehmen. Dies geht heute bis zur Nutzung des World-Wide-Web als Frontend, über das dem Endkunden Buchungssysteme angeboten werden. Hier wird nicht nur wegen der einfacheren Bedienung für Gelegenheitsbenutzer, sondern oft auch aus Marketingüberlegungen Wert auf eine optisch ansprechende Oberfläche gelegt.

Die sog. GUIfizierung von Host-Transaktionen erfolgt durch den Entwurf neuer GUI-Masken, die sich meist stark an den bestehenden Dialogabläufen orientieren, jedoch bessere Hilfefunktionalität für Gelegenheitsbenutzer bieten. Die Dialogsteuerung [Den91] kann sogar auf dem Host bleiben (Abbildung 4). Eine verbreitete Variante ist, daß die Client-Anwendung für den Host ein zeichenorientiertes Terminal emuliert. Datentypprüfungen können auf dem Client erfolgen. Die Funktionalität der Anwendung kann um graphische Selektionsmöglichkeiten erweitert werden[7].

Die Varianten zu dieser Strategie sind zahlreich. Reid [Rei95] schildert Ansätze, mit denen man versuchen kann, einen Teil eines Objektmodells auf dem Client zu installieren und mit Transaktionen zu ver- und entsorgen. Diese Ansätze sind jedoch komplex und auch was die Transaktionssicherheit angeht bedenklich. Es ist nicht opportun, ein Anwendungskernobjekt auf einem Client mit mehren *Conversational Transactions* auf den Server zu sichern, da diese jeweils meist auch einer Datenbanktransaktion auf dem Host entsprechen. Damit sind auf dem Client keine langen Benutzertransaktionen möglich, wenn nicht spezielle Vorkehrungen[8] getroffen werden. Statt dessen bietet sich eher die Strategie „Objektorientierte Datenintegration“ an.

Die Vorteile eine GUIfizierung sind auf den ersten Blick bestechend. Die Performance ist nicht schlechter als bei einem zentralen Host-System. Die Migrationskosten sind gering. Fertige Kapselungsmechanismen für Transaktionen finden sich heute zum Beispiel schon in Smalltalk-Programmierumgebungen. Die Entwicklung kann voll entkoppelt vom normalen Betrieb erfolgen. Basis sind getestete, bewährte Anwendungen.

Die Nachteile sind erst auf den zweiten Blick sichtbar. Mindestens durch dezentrale Typprüfungen auf einem Client entsteht redundanter, fachlicher Code. Es entsteht ferner ein redundantes Hilfesystem. Werden auf einem Client noch An-

---

[6] Zu finden ist dieser Ansatz häufig in der Touristikbranche. Denkbar ist er aber auch in anderen Branchen. Die betriebswirtschaftliche Forderung nach der Verknüpfung von Wertschöpfungsketten [Por86] ist nicht neu. Sie kann durch das dargestellte Strategiemuster oft schnell erreicht werden.

[7] Hier handelt es sich dann oft um eine Kombination mit einem Offline-System, das erweiterte Stammdaten zum Lesen zur Verfügung stellt.

[8] Man kann durch einen speziellen Paketierungsmechanismus mehrere Aktionen, die auf einem Host ausgeführt werden sollen, zu langen Transaktionen bündeln [Kel+96, S. C644-16-17]. Dies setzt jedoch eine aufwendigere Infrastruktur voraus und ist bei vorhandenen Transaktionen auch nicht in jedem Fall machbar.

wendungskernfunktionen repliziert, entsteht noch mehr redundanter Code. Es kann weiter nicht behauptet werden, daß das dargestellte Vorgehen zu einer klaren objektorientierten Architektur hinführt. Die Dialogsteuerung auf dem Host ist mit dem in OO-Architekturen bevorzugt einzusetzenden MVC-Paradigma [Kra+88] nicht vereinbar. Sie ist am besten noch mit einer X Windows Architektur [Sch+86] zu vergleichen, ohne jedoch deren Vorteile der Plattformunabhängigkeit, der zentralen Anwendungslogik und der verteilten Präsentation zu besitzen. Versucht man, Anwendungskernobjekte auf einem Client zu implementieren, deren Instanzvariablen aus *Conversational Transactions* ge- und entladen werden (zum Prinzip siehe [Rei95] und Abbildung 5), so erhält man extrem komplexe und nicht immer zuverlässige Lösungen, die noch dazu kaum Chancen zu einer späteren Migration in Richtung Redundanzfreiheit und CORBA bieten. Der einzige Vorteil einer solchen Strategie ist, daß der Persistenzmechanismus später ausgewechselt und die Oberfläche schon über MVC mit den Anwendungskern-Objekten gekoppelt werden kann.

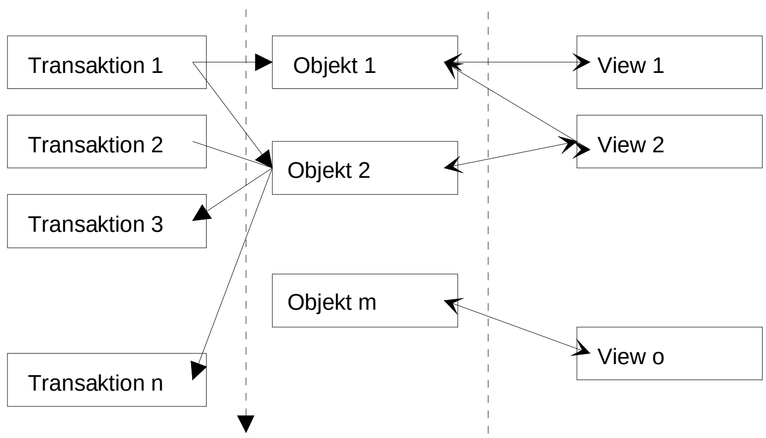

Abbildung 5: Ver- und Entsorgung von Attributen aus Transaktionen: Objekt 2 wird aus den Transaktionen 1 und 2 mit Attributen versorgt und durch die Transaktionen 3 und n in die Server-Datenbasis gesichert. Da zwischen Transaktionen und Objekten i.a. eine n:m Beziehung besteht, ist einsichtig, daß eine solche Datenversorgung komplex ist. Dazu kommt die fehlende Transaktionssicherheit, da die Transaktionen 1 bis n auf einem Host jeweils als eigene Transaktionen behandelt werden.

5 Migration mit CORBA

Die Strategie *„Migration mit CORBA"* steht für den Ansatz, Altsysteme mittels sogenannter Kapseln (*Wrapper* [Gra95-S.41ff, Lyo95, Mul95, Win95]) in Objekte umzuwandeln. Diese Objekte können dann unter Zuhilfenahme von *Object Request Brokern* [OMG95, Bus+96] in einer heterogenen verteilten Umgebung von neuen objektorientierten Anwendungen benutzt werden. Die Kapsel wird in der Regel um den Anwendungskern des Altsystems gelegt, obwohl die Technologie auch für die Kapselung auf der Maskenebene (Strategie „*Alte Transaktionen* *mit neuem GUI")* oder der Datenebene (Strategie *„Objektorientierte Datenintegration")* eingesetzt werden kann. CORBA steht hier nur stellvertretend für eine „*Object Broker Architektur*". Eine Diskussion CORBA versus OLE soll hier nicht geführt werden[9].

Das Problem, das gelöst werden soll, ist die mittel- und langfristige Koexistenz von Altsystemen und neuen objektorientierten Anwendungen in verteilten (und möglicherweise heterogenen) Umgebungen. Nach einer Phase erster objektorientierter Pilotanwendungen, die eine nur geringe Integration mit Altsystemen benötigen (wie etwa im Falle von Offline-Systemen) stellt sich meist das Problem, neue objektorientierte Anwendungen mit existierenden Anwendungen koppeln zu müssen. Ist man bei dieser Kopplung unvorsichtig und realisiert sie mit Funktionsaufrufen (etwa über *Remote Procedure Calls*), so breiten sich nicht-objektorientierte Bereiche großflächig in neuen Anwendungen aus. Das zugrundeliegende Problem kann auch als der Versuch beschrieben werden, die in den Altsystemen steckende große Investition durch Verlängerung ihrer „Lebenszeit" in eine Welt verteilter Objekte besser zu amortisieren.

Die in einem monolithischen Altsystem hoffentlich vorhandene interne Schnittstelle zwischen dem Anwendungskern und der Benutzungsschnittstelle wird zunächst freigelegt. Hat das Altsystem mindestens eine Schichtenarchitektur [Den91] und einen klar separierbaren Anwendungskern, so ist hier vergleichsweise wenig zu tun. Anderenfalls muß mittels eines Reengineering-Schrittes erst eine echte technische Auftrennung zwischen Anwendungskern und Benutzungsschnittstelle erfolgen. Anschließend wird die Schnittstelle des Anwendungskerns mittels Tools oder notfalls auch manuell dazu verwendet, Kapselobjekte aufzubauen. Kapselobjekte genügen an ihren Schnittstellen dem CORBA-Standard. Die Schnittstellen werden mit IDL (*Interface Definition Language* [OMG95]) beschrieben. Das Schnittstellenobjekt leitet alle Aufrufe an den Anwendungskern des Altsystems weiter. Eine eigene Datenhaltung oder fachliche Logik ist in einem Kapselobjekt in der Regel nicht notwendig. Das Altsystem kann nun mittels *Object Request Broker* verwendet werden, als bestünde es aus einem oder mehren Objekten. Es kann parallel dazu auch weiterhin über seine eigene Benutzungsschnittstelle in Betrieb bleiben.

Varianten

Varianten sind unter anderem die schon erwähnte Möglichkeit, mittels CORBA nicht den Anwendungskern, sondern die Transaktionen der Benutzungsschnittstelle oder die Datenhaltung zu kapseln. CORBA wird hier allerdings nicht im

---

[9] In [Orf+96] ist detailliert beschrieben, aus welchen Gründen CORBA 2.0 dem heutigen OLE mit weitem Abstand vorzuziehen ist.

objektorientierten Sinn, sondern als Remote Procedure Call (RPC) genutzt, da in der Regel keine Objektinstanzen erzeugt werden. Darüber hinaus gibt es eine Reihe von Varianten bei der Kapselung des Anwendungskerns eines Altsystems, die sich im Granulat der Kapselung unterscheiden.

- **Schnittstellenobjekte:** Die oben beschriebene Vorgehensweise kann im Extremfall bedeuten, daß die funktionale Schnittstelle des Anwendungskerns eines Altsystems komplett zu einer Methodenschnittstelle einer oder weniger Anwendungskernkapseln (Schnittstellenobjekte) wird. Dabei wird lediglich die durch CORBA vorhandene Infrastruktur für den Aufruf verteilter Objekte genutzt. Die Grundidee der Objektorientierung, nämlich die Bündelung und Kapselung fachlich zusammenhängender Daten und Methoden wird verletzt. CORBA degeneriert hier zum RPC-Mechanismus. Ein Kapselungsobjekt kann in der Regel eine sehr große Zahl von Funktionen des Altsystemanwendungskerns enthalten, die miteinander nur in losem oder gar keinen fachlichen Zusammenhängen stehen.
- **Statische Business Objects ohne Instanzen:** Ein strukturierter Anwendungskern besteht meist aus größeren disjunkten Einheiten (Module oder Subsysteme), die mittels entsprechender Anwendungskernkapseln getrennt voneinander als Objekte angeboten werden können. Die resultierenden Objekte sind statischer Natur und entsprechen in etwa den Domains in der objektorientierten Analyse [Boo96] bzw. sogenannten *"Business Objects"* [BOM95]. In diesem Ansatz werden jedoch immer noch nicht dynamisch neue Objektinstanzen erzeugt. Die Kapseln [Klö+95] bilden eine statische Strukturierung des Anwendungskerns ab. Man arbeitet also mit Metaklassen und adressiert Objekte nie direkt als Instanzen sondern über die Metaklasse mit einem fachlichen Schlüssel.
- **Persistente Objekte mit Instanzen:** Die eigentliche Kapselung des Anwendungskerns in objektorientierter Form bieten persistente[10], fachliche Objekte. Anstelle eines Objekts "Partnerverwaltung" wird man in diesem Ansatz einzelne Objekte für jeden Kunden, bzw. Partner gewinnen. Eine solche Kapselung ist in einem nicht objektorientiert entworfenen System ohne massive Restrukturierung kaum möglich[11]. Wenn aber Analyse und Design des Altsystems objektorientiert vorgenommen wurden und lediglich in der Realisierung konventionell vorgegangen wurde, sind fachliche persistente Objekte durchaus zu gewinnen. Fachliche persistente Objekte bieten echte Kapselung von Daten

---

[10] Zum Begriff siehe [Atk+83]. Zu Arten siehe z.B. [Sou94].

[11] Klösch und Gall beschreiben in ihrem Buch „Objektorientiertes Reverse Engineering" [Klö+95] den Aufwand, der zu treiben ist, wenn man auch nur die Stufe *„Statische Business Objects ohne Instanzen"* erreichen will. Die Stufe *„Persistente Objekte mit Instanzen"* wird in der Praxis kaum noch durch normales Reengineering erreicht, sondern eher durch ein teilweises Neuschreiben von Anwendungen, dem allerdings eine gründliche Analyse der Altsysteme [Ebe+95, Klö+95] vorausgehen sollte.

durch die Verarbeitungslogik. Insbesondere in Bezug auf Code-Wiederverwendung und die Durchsetzung von Integritätsbedingungen ist dies der eigentlich objektorientierte Weg der Kapselung.

Vor- und Nachteile

Die großen Vorteile der Migration mit CORBA nach der Variante „*Schnittstellenobjekte*“ liegen in geringen Migrationskosten und einer guten Entkopplung. Dies gilt aber nur, solange kein Reengineering vorgenommen werden muß. Darüber hinaus sind die Restrukturierungskosten gering, weil die Kapselung ja gerade das Ziel hatte, Altsysteme unangetastet zu lassen. Diesen Vorteil genießt man aber nur bei gut stukturierten Altsystemen. Die Entkopplung in Bezug auf Dialoge und Datenhaltung des Altsystems ist hier gut zu nennen. Während die Dialoge im wesentlichen unbehindert weiterentwickelt werden können gilt dies nicht für den Anwendungskern, da er doppelt genutzt wird – von Altsystemteilen und Wrappern. Der Vorteil der Entkopplung geht jedoch völlig verloren, wenn für die Varianten *„Statische Business Objects ohne Instanzen“* oder *„Persistente Objekte mit Instanzen“* eine Restrukturierung vorgenommen wird. Diesen Fall kann man auch als Restrukturierung und Systemerweiterung betrachten. Er unterscheidet sich nur noch unwesentlich von einem kompletten Reengineering.Als wesentlicher Nachteil muß heute eindeutig noch die Performance von CORBA-Implementierungen, aber natürlich auch der Overhead der eigentlichen Kapselung angesehen werden. Wenn eine bislang programminterne Schnittstelle zwischen einem Dialog oder einem Batch und dem Anwendungskern nun zu einem Client/Server-Schnitt wird, sind Performanceprobleme vorprogrammiert. Hier gilt es, tragfähige Kompromisse in Bezug auf das Kapselungsgranulat zu finden. Ein weiterer Nachteil betrifft die Kriterien Entkopplung und Klarheit der Architektur. Kapselung kann dazu führen, daß schlecht strukturierte Systeme für viele weitere Jahre zementiert und Neuentwicklungen kompliziert werden.

Ein möglicher Ausweg aus diesen Dilemmata besteht darin, eine grobe objektorientierte Analyse für alle Anwendungen eines Unternehmens durchzuführen und auf dieser Basis festzustellen, für welche Altsysteme eine Kapselung, und für welche anderen ein Reengineering oder eine komplette Neuerstellung sinnvoller sind [Ebe+95].

6 Objektorientierte Datenintegration

Bei der Strategie „*objektorientierte Datenintegration*“ werden die Anwendungskernobjekte neu erstellter Systeme persistent gemacht, indem man dieselbe Datenbasis verwendet, auf der auch bestehende Anwendungen operieren (siehe Abbildung 6).

Alte Anwendungen und objektorientiert realisierte Komponenten operieren dabei auf denselben Datenbanken und dort auch teilweise auf identischen Tabellen. Vor einem Programmierer, der nur Anwendungskernobjekte neuer Systemteile verwendet, kann die Datenherkunft weitgehend versteckt werden [Col+95, Kel+96]. Die Abbildung von Objekten auf relationale oder hierarchische Altdaten wurde in der Literatur wesentlich ausführlicher untersucht (zum Beispiel bei [Bur94, Hah+95, Kim95, Pre+94, Wal+95]) als die bisher vorgestellten Strategien. Die Untersuchungen erfolgten jedoch vor allem im Hinblick auf die technische Machbarkeit und erst in zweiter Linie im Hinblick auf den sinnvollen Einsatz einer solchen Strategie in einer Migration zu objektorientierten Architekturen.

Das mit der Strategie zu lösende Problem ist meist, daß man objektorientierte Anwendungen bauen möchte, die einen hohen Anteil an neuen Objekten haben, die noch in keinem Altsystemteil vorhanden sind. Die Migration zu einem objektorientierten Datenhaltungssystem [ODMG93] soll aber noch nicht erfolgen, da oft gerade erst in relationale Technologie investiert wurde. Die Architektur kommt auch für ein schrittweises, objektorientiertes Reengineering von Host-Anwendungen in Frage.

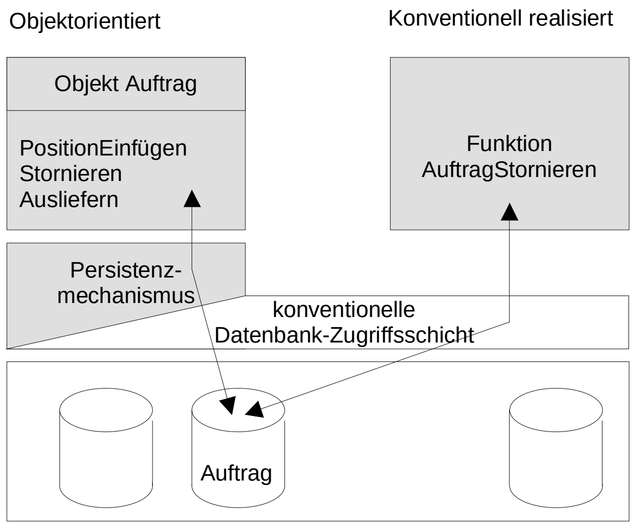

Abbildung 6: Objekte und konventionelle Funktionen verwenden dieselben Datenbanken

Die Lösungen und Produkte, die bei dieser Strategie eingesetzt werden können, sind enorm vielfältig. Man kann für den Weg zu einer solchen Lösung wieder eigene Strategien und eine Rahmenarchitektur (siehe Abbildung 7) angeben [Kel+96]. Das Spektrum reicht dabei von einer Kombination mit Application Frameworks [Cot+95, Kel+95, Lip95] über objektorientierte Zugriffsschichten [Hah+95] bis zu objektorientierten Datenbanken mit Gateways und dem Einsatz von föderierten SQL3-Datenbanken [Kim95, Sto94]. Bei jeder der Lösungen sind meist sogar noch verschiedene Client/Server-Schnitte möglich.

Die Vorteile der Strategie *„Objektorientierte Datenintegration“* liegen vor allem in der Klarheit der Architektur, die es ermöglicht, einmal erstellte Anwendungskernobjekte später auch mit einer objektorientierten Datenbank persistent und mit einem Object Request Broker allgemein verfügbar zu machen. Entkopplung von Altanwendungen kann durch Maßnahmen auf Datenbankebene sichergestellt werden. Die Migrationskosten ergeben sich als Folge der jeweiligen Produktentscheidung. Individuelle Lösungen, die durch spezielle Systemplattformen erzwungen werden, sind dabei wesentlich teurer als Frameworks und objektorientierte Datenbankprodukte mit Gateways zu relationalen Schnittstellen.

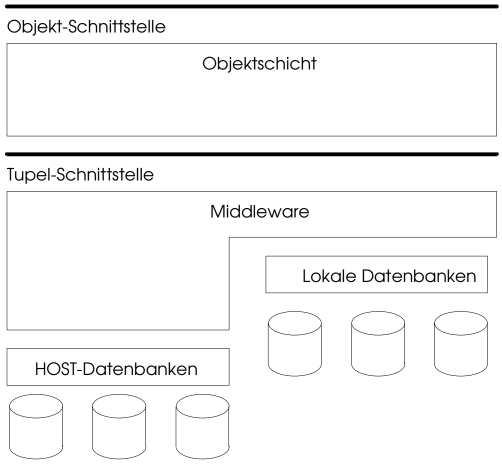

Abbildung 7: Architekturrahmen für die Strategie "Objektorientierte Datenintegration" [Kel+96]

Die Aussagen über die Nachteile sind ebenfalls stark von der eingesetzten Lösungsvariante abhängig. Auf die Performance muß besonders geachtet werden. Redundante Anwendungskernobjekte sind eher die Ausnahme. Eine Verletzung der Architektur ergibt sich durch die Verletzung der Datenkapselung, wenn alte und neue Anwendungen schreibend auf dieselben Daten zugreifen.

7 Zusammenfassung

Die Diskussion der einzelnen Ansätze hat gezeigt, daß es *eine* optimale Migrationsstrategie nicht gibt. Es gibt aber Muster, aus denen sich eine angepaßte Strategie für jedes Unternehmen zusammensetzen läßt. Die konkreten Ansätze ergeben sich aus den individuellen Gegebenheiten des Unternehmens, das zu einer objektorientierten Systemarchitektur migrieren möchte. Je nach den Gegebenheiten sind auch mehrere Strategiemuster kombinierbar und anwendbar. Das Ziel sollte langfristig eine durchgängig objektorientierte Architektur sein. Der Weg zu diesem Ziel kann lang sein und mehrere gut geplante Schritte erfordern. Bei der Planung der einzelnen Schritte kann man sich der hier vorgestellten Strategien bedienen.

Danksagung

Diese Arbeit entstand im Rahmen des Verbundprojektes „ENTSTAND – Standardarchitektur für Informationssysteme“ das vom BMBF gefördert wird. Dafür möchten wir uns hier bedanken.

Literatur

[Ale79] **Christopher Alexander**: *The Timeless Way of Building*, Oxford University Press 1979.

[Atk+83] **M. P. Atkinson, P. J. Bailey, K. J. Chisholm, W. P. Cockshott, R. Morrison**: *An Approach to Persistent Programming*, The Computer Journal, 26(4), 1983.

[BOM95] **Object Management Group, BOMSIG**: *Business Applications Architecture*, OMG 1995.

[Boo96] **Grady Booch**: *Object Solutions – Managing the Object Oriented Project*, Addison-Wesley 1996.

[Bur94] **D. K. Burleson**: *Practical Application of Object-Oriented Techniques to Relational Databases*, John Wiley & Sons 1994.

[Bus+96] **F. Buschman, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal**: *Pattern Oriented Software Architecture*, Wiley 1996.

[Cat94] **Rick G. G. Cattell**: *Object Data Management*, Addison-Wesley 1994.

[Col+95] **Jens Coldewey, Wolfgang Keller**: *Objektorientierte Datenzugriffe auf dem VAA Datenmanager*, GDV, Bonn 1995.

[Cop+95] **James O. Coplien, Douglas C. Schmidt (Hrsg.)**: *Pattern Languages of Program Design*, Addison-Wesley 1995.

[Cot+95] **Cotter, Pottel**: *Inside Taligent Technology*, Addison Wesley 1995.

[Den91] **Ernst Denert**: *Software-Engineering*, Springer Verlag 1991.

[Ebe+95] **Klaus Eberhardt, Sebastian Kutscha**: *Veraltete Systeme hemmen Innovation – Altsystem-Abhängigkeit ist wichtiger als High-Tech-Themen*, Computerwoche CW-Extra, 17. Februar 1995.

[GOF95] **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides**: *Design Patterns, Elements of Reusable Object Oriented Software*, Addison-Wesley 1995.

[Gol+95] **Adele Goldeberg, Kenneth Rubin**: *Succeeding With Objects*, Addison-Wesley 1995.

[Gra+93] **Jim Gray, Andreas Reuter**: *Transaction Processing, Concepts and Techniques,* Morgan Kaufmann Publishers 1993.

[Gra95] **Ian Graham**: *Migrating to Object Technology*, Addsion-Wesley 1995.

[Hah+95] **Wolfgang Hahn, Fridtjof Toennissen, Andreas Wittkowski**: *Eine objektorientierte Zugriffsschicht zu relationalen Datenbanken*, Informatik Spektrum 18(3), Seite 143–151.

[Han95] **Wolf-Rüdiger Hansen (Hrsg.)**: *Client/Server-Architektur*, Addison-Wesley 1995.

[Har+94] **H. J. Hartmann, A. Lannes**: *Einstieg in eine verteilte DV-Landschaft – Reengineering des Vertriebssystems bei der NUR-Touristic GmbH*, Computerwoche Special Nr. 4, 21. Oktober 1994, pp. 24–28.

[Kel+95] **Michael Keller, Thomas Stalzer**: *Ein Erfahrungsbericht über den Einsatz von VisualAge und Vaser*, OBJEKTspektrum, 2(4), Juli/August 1995.

[Kel+96] **Wolfgang Keller, Christian Mitterbauer, Klaus Wagner**: *Objektorientierte Datenintegration über mehrere Technologiegenerationen*, Proceedings ONLINE, Kongress VI, Hamburg 1996.

[Kim95] **Won Kim (Hrsg.)**: *Modern Database Systems,* ACM Press 1995.

[Klö+95] **Rene Klösch, Harald Gall**: *Objektorientiertes Reverse Engineering – Von klassischer zu objektorientierter Software*, Springer Verlag 1995.

[Kra+88] **G. E. Krasner, S. T. Pope**: *A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80*, Journal of Object Oriented Programming, August/September 1988.

[Lip95] **Peter Lipps**: *Enterprise Objects Framework – Fachspezifische Objekte in Open Step*, OBJEKTspektrum, 2(5), September/Oktober 1995.

[Lyo95] **Daniel Lyons**: *Wrapping Up Legacy Code*, InfoWorld, June 26, 1995 17(26), pp 59(2).

[Mey95] **Bertrand Meyer**: *Object Success*, Prentice Hall 1995.

[Mul95] **Diane E. Mularz**: *Pattern Based Integration Architectures*, in **James O. Coplien, Douglas C. Schmidt (Hrsg.)**: *Pattern Languages of Program Design*, Addison-Wesley 1995, pp 441-452.

[ODMG93] **Rick G. G. Cattell (Ed.) et. al.**: *Object Database Standard (ODMG 93),* Morgan Kaufmann Publishers 1993.

[OMG95] **Object Management Group**: *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, OMG, Framingham, MA July 1995.

[Orf+94] **Robert Orfali, Dan Harkey**: *Client/Server Survival Guide,* Van Nostrand Reinhold 1994.

[Orf+96] **R. Orfali, D. Harkey, J. Edwards**: *The Essential Distributed Objects Survival Guide*, Wiley, 1996.

[Por86] **Michael E. Porter***: Wettbewerbsvorteile, Spitzenleistungen erreichen und behaupten*, Frankfurt a. M. 1986.

[Pre+94] **William Premerlani, Michal R. Blaha**: *An Approach for Reverse Engineering of Relational Databases,* Communications of the ACM, May 1994, p42(9).

[Rei95] **Glen Reid:** *Interfacing to Legacy Systems*, Object Magazine 5(6), October 1995, pp 46–51.

[Sch+86] **R. W. Scheifler, J. Gettys:** *The X Window System*, ACM Transactions on Graphics 5(2), pp. 79-109, April 1986.

[Sou94] **Jiri Soukup**: *Taming C++ – Pattern Classes and Persistence for large Projects*, Addison-Wesley 1994.

[Sto94] **Michael Stonebreaker**: *Object-Relational Database Systems*, Proceedings ObjectWorld 94, London 1994.

[Wal+95] **Kim Walden, Jean-Marc Nerson**: *Seamless Object Oriented Software Architecture*, Prentice Hall 1995.

[Win95] **Paul Winsberg** , *Legacy Code: Don't Bag It, Wrap It,* Datamation, May 15, 1995 41(9) pp36(4).